# SQL Programming 1

- The SELECT-FROM-WHERE Structure
- Single Relation Queries
  - ORDER BY
  - LIKE
  - IS (NOT) NULL
  - DISTINCT
- Aggregation Queries
  - Grouping
  - Having
- Oracle SQL*Plus

Readings: Section 6.1 and 6.4 of Textbook.
The SQL*Plus tutorial by R. Holowczak

# Why SQL?

- SQL is a very-high-level language.
  - Say "what to do" rather than "how to do it."
  - Avoid a lot of data-manipulation details needed in procedural languages like C++ or Java.
- Database management system figures out "best" way to execute a query.
  - Called "query optimization."

# SELECT-FROM-WHERE

An SQL query consists of three components: a SELECT clause, a FROM clause and an optional WHERE clause .

SELECT desired attributes
FROM one or more tables
WHERE conditions on tuples of the tables

# Our Running Example

The Village Cinema database:

Movie(mvID, Title, Rating, Rel_date, Length, Studio )

Classification(mvID*, Genre)

Cast(mvID*, Actor)

Direct(mvID*, Director)

All our SQL queries will be based on the Village Cinema database schema.

# Example

- What movies are produced by Roadshow?

```
SELECT title
FROM Movie
WHERE studio = 'Roadshow';
```

- The answer is a relation with a single attribute, Title, and tuples of the title of each movie by Roadshow.

```
TITLE
-----------------------------------------------
Coco Avant Chanel
Harry Potter and the Half-Blood Prince
```

# The FROM-WHERE-SELECT Process

- FROM: Imagine a pointer visiting each tuple of the relation in FROM.
- WHERE: Check if the "current" tuple satisfies the WHERE clause.
- SELECT: If so, compute the attributes or expressions of the SELECT clause using the components of this tuple.

# The FROM-WHERE-SELECT Process …

- FROM movie:
  - Loop over rows in the Movie table.
- WHERE studio = 'Roadshow':
  - for each row of the Movie table, check if the condition holds --- its value for Studio is Roadshow.
- SELECT title:
  - For each resultant tuples from the WHERE Clause, output the Title.

# The FROM-WHERE-SELECT process ...

**Movie**

| mvID | Title | Rating | Rel_date | Length | Studio |
|---|---|---|---|---|---|
| 1 | Angels & Demons | M | 14-05-2009 | 138 | Sony Pictures |
| 2 | Coco Avant Chanel | PG | 25-06-2009 | 108 | Roadshow |
| 3 | Harry Potter and the Half-Blood Prince | M | 15-07-2009 | 153 | Roadshow |
| 4 | The Proposal | PG | 18-06-2009 | 107 | Disney |
| 5 | Ice Age: Dawn of the Dinosaurs | PG | 01-07-2009 | 94 | 20th Century Fox |



| Title |
|---|
| Coco Avant Chanel |
| Harry Potter and the Half-Blood Prince |

```
Select title
From movie
Where studio='Roadshow';
```

# SELECT *

- * in the SELECT clause stands for "all attributes of the relations in the FROM clause."
- Example:

```
SELECT *
FROM Movie
WHERE studio = 'Roadshow';
```

# Result of Query

**Movie**

| mvID | Title | Rating | Rel_date | Length | Studio |
|---|---|---|---|---|---|
| 1 | Angels & Demons | M | 14-05-2009 | 138 | Sony Pictures |
| 2 | Coco Avant Chanel | PG | 25-06-2009 | 108 | Roadshow |
| 3 | Harry Potter and the Half-Blood Prince | M | 15-07-2009 | 153 | Roadshow |
| 4 | The Proposal | PG | 18-06-2009 | 107 | Disney |
| 5 | Ice Age: Dawn of the Dinosaurs | PG | 01-07-2009 | 94 | 20th Century Fox |

**Movie**

| mvID | Title | Rating | Rel_date | Length | Studio |
|---|---|---|---|---|---|
| 2 | Coco Avant Chanel | PG | 25-06-2009 | 108 | Roadshow |
| 3 | Harry Potter and the Half-Blood Prince | M | 15-07-2009 | 153 | Roadshow |

Now, the result has each of the attributes of Movie.

# SELECT *Expressions*

- Any expression that makes sense can appear as an element of a SELECT clause.
- Example: Find the length of movies in hours.

```
SELECT title, length/60
FROM movie
```

| TITLE | LENGTH/60 |
|---|---|
| Angels and Demons | 2.3 |
| Coco Avant Chanel | 1.8 |
| Harry Potter and the Half-Blood Prince | 2.55 |
| The Proposal | 1.78333333 |
| Ice Age: Dawn of the Dinosaurs | 1.56666667 |

# Aliases: SELECT ... AS ...

- Attributes can be renamed using AS .... This is often used for producing reports.

```
SELECT title, length/60 as LenInHours
FROM movie;
```

| TITLE | LENGINHOURS |
|---|---|
| Angels and Demons | 2.3 |
| Coco Avant Chanel | 1.8 |
| Harry Potter and the Half-Blood Prince | 2.55 |
| The Proposal | 1.78333333 |
| Ice Age: Dawn of the Dinosaurs | 1.56666667 |

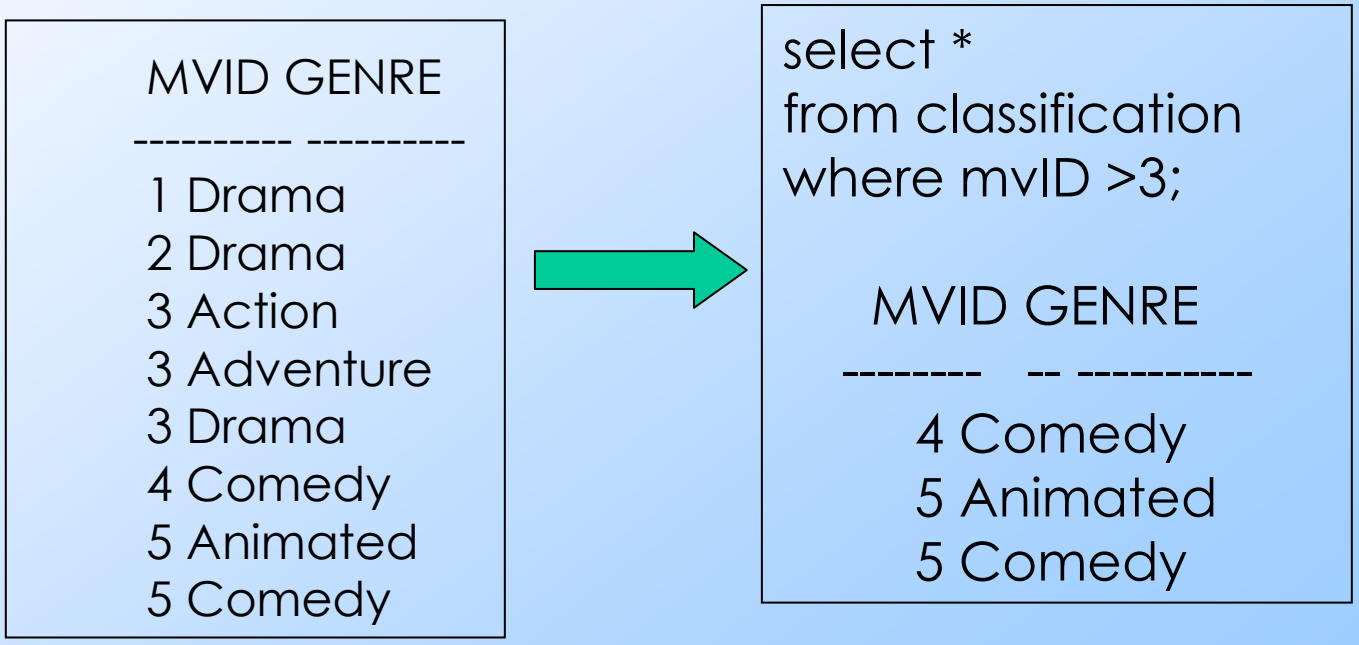
# Select Constants

```
SELECT 'I like', title
FROM Movie
WHERE studio = 'Roadshow';
```

```
'ILIKE TITLE
------ ----------------------------------------
I like Coco Avant Chanel
I like Harry Potter and the Half-Blood Prince
```

# The WHERE Clause

- In the WHERE clause, an expression is evaluated to a boolean value (TRUE or FALSE). Tuples are selected only if the WHERE expression is evaluated to TRUE.
- Comparisons result in boolean values.
  - =, <>, <, >, <=, >=, !=.
- Example: Find the genre of movies whose mvID is >3.



# The WHERE Clause ...

- Many other operators also result in boolean values. These more complex operators will be explained later.
  - LIKE, IS NULL, IS NOT NULL.
- Logical operators combine boolean expressions into a complex condition.
  - NOT, AND, OR.

# Example: Complex Condition

- Find the title of movies produced by Roadshow and with rating 'PG'.

```
SELECT title
FROM Movie
WHERE studio='Roadshow' AND rating='PG';
```

# String Patterns for Comparison

- A condition can compare a string to a pattern by:
  - <Attribute> LIKE <pattern> or <Attribute> NOT LIKE <pattern>
- Pattern is a quoted string with
  - %: any string;
  - _: any character.

# Example: LIKE

- Find the movies whose title has the string "Age". Return their title.

```
SELECT title
FROM Movie
WHERE title LIKE '%Age%';
```

# NULL Values

- Tuples in SQL relations can have NULL as a value for one or more components.
- Meaning depends on context. Two common cases:
  - *Missing value* : e.g., we do not know the length of movie "Angels and Demons".
  - *Inapplicable* : e.g., the value of attribute spouse for an unmarried person.

# NULL Comparison

- WHERE … IS NULL:
  - Check if an attribute value equals to NULL.

Example: Find the movies where the production studio information is missing.

```
SELECT *
FROM Movie
WHERE studio IS NULL
```

# NULL Comparison …

- WHERE … IS NOT NULL.
  - Check if an attribute does not take NULL value.

Exercise: Explain the following query in English.

```
SELECT *
FROM Movie
WHERE studio IS NOT NULL
```

# Exercises

Write an SQL query to find the mvID of movies that have either "Tom Hanks" or "Audrey Tautou".

| MVID | ACTOR |
|---|---|
| 1 | Tom Hanks |
| 2 | Alessandro Nivola |
| 2 | Audrey Tautou |
| 2 | Benolt Poelvoorde |
| 2 | Marie Gillain |
| 3 | Daniel Radcliffe |
| 3 | Emma Watson |
| 3 | Rupert Grint |
| 4 | Betty White |
| 4 | Malin Akerman |
| 4 | Mary Steenburgen |
| 4 | Ryan Reynolds |
| 4 | Sandra Bullock |
| 5 | Chris Wedge |
| 5 | Denis Leary |
| 5 | John Leguizamo |
| 5 | Queen Latifah |
| 5 | Ray Ramono |

# Exercises ...

The query given below is meant to find the mvID of movies that have "Marie Gillain" and "Audrey Tautou". Will the query do the job? What's the result for the following queries?

| MVID | ACTOR |
|---|---|
| 1 | Tom Hanks |
| 2 | Alessandro Nivola |
| 2 | Audrey Tautou |
| 2 | Benolt Poelvoorde |
| 2 | Marie Gillain |
| 3 | Daniel Radcliffe |
| 3 | Emma Watson |
| 3 | Rupert Grint |
| 4 | Betty White |
| 4 | Malin Akerman |
| 4 | Mary Steenburgen |
| 4 | Ryan Reynolds |
| 4 | Sandra Bullock |
| 5 | Chris Wedge |
| 5 | Denis Leary |
| 5 | John Leguizamo |
| 5 | Queen Latifah |
| 5 | Ray Ramono |

```
select mvID
from cast
where actor='Marie Gillain'
        AND actor='Audrey Tautou';
```

Notes: The WHERE clause can run across several lines. So there are not any grammar errors.

# DISTINCT: Removing Duplicates

- What are the genres for movies?

  SELECT genre

  FROM Classification;

- Each genre is listed as many times as there are movie-genre pairs.

| GENRE |
| --- |
| Drama |
| Drama |
| Action |
| Adventure |
| Drama |
| Comedy |
| Animated |
| Comedy |

# DISTINCT: Removing Duplicates

- What are the (distinct) genres for movies?

  SELECT **DISTINCT** genre

  FROM Classification;

- Each different genre is listed only once.

| GENRE |
| --- |
| --- |
| Adventure |
| Animated |
| Action |
| Comedy |
| Drama |

# ORDER BY: Ordering Output

- By default ORDER BY sort output into asending order. But can use ORDERY BY ... DESC to sort into descending order.
- Example:

```
select mvID, title, length
from movie
order by length desc
```

| MVID | TITLE | LENGTH |
|---|---|---|
| 3 | Harry Potter and the Half-Blood Prince | 153 |
| 1 | Angels and Demons | 138 |
| 2 | Coco Avant Chanel | 108 |
| 4 | The Proposal | 107 |
| 5 | Ice Age: Dawn of the Dinosaurs | 94 |

# Aggregations

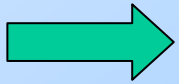
- SUM, AVG, COUNT, MIN, and MAX can be applied to a column in a SELECT clause to produce that aggregation on the column.
- Also, COUNT(*) counts the number of tuples.

# Example: Aggregation

- Overall aggregation: Aggregate all tuples in a table.
- Example: What's the average length of movies overall?

```
SELECT AVG(length)
FROM Movie;
```

| MVID | LENGTH |
|---|---|
| 1 | 138 |
| 2 | 108 |
| 3 | 153 |
| 4 | 107 |
| 5 | 94 |



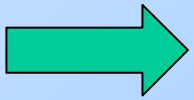
| AVG(LENGTH) |
|---|
| 120 |

# Selective Aggregation

- Aggregates the tuples selected by the WHERE clause.
- Example: What is the average length of movies produced by Roadshow?

```
SELECT AVG(length)
FROM Movie
WHERE studio='Roadshow';
```

| MVID | LENGTH | STUDIO |
|---|---|---|
| 2 | 108 | Roadshow |
| 3 | 153 | Roadshow |



| AVG(LENGTH) |
|---|
| 130.5 |

# Eliminating Duplicates in an Aggregation

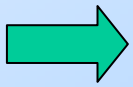
- Use DISTINCT inside an aggregation.
- Example: How many studios are there in total?

```
SELECT COUNT(DISTINCT studio)
FROM Movie;
```

# GROUP BY: Grouping Tuples

- GROUP BY groups tuples -- possibly selected by a WHERE clause – into groups for aggregation.
  - Each group of tuples have unique values for the group-by attribute list.
- Example: How many movies are there for each genre?

```
 MVID GENRE
---------- ----------
         1 Drama
         2 Drama
         3 Action
         3 Adventure
         3 Drama
         4 Comedy
         5 Animated
         5 Comedy
```



```
select genre, count(*)
from classification
group by genre;

GENRE      COUNT(*)
---------- ----------
Adventure           1
Animated            1
Action              1
Comedy              2
Drama               3
```

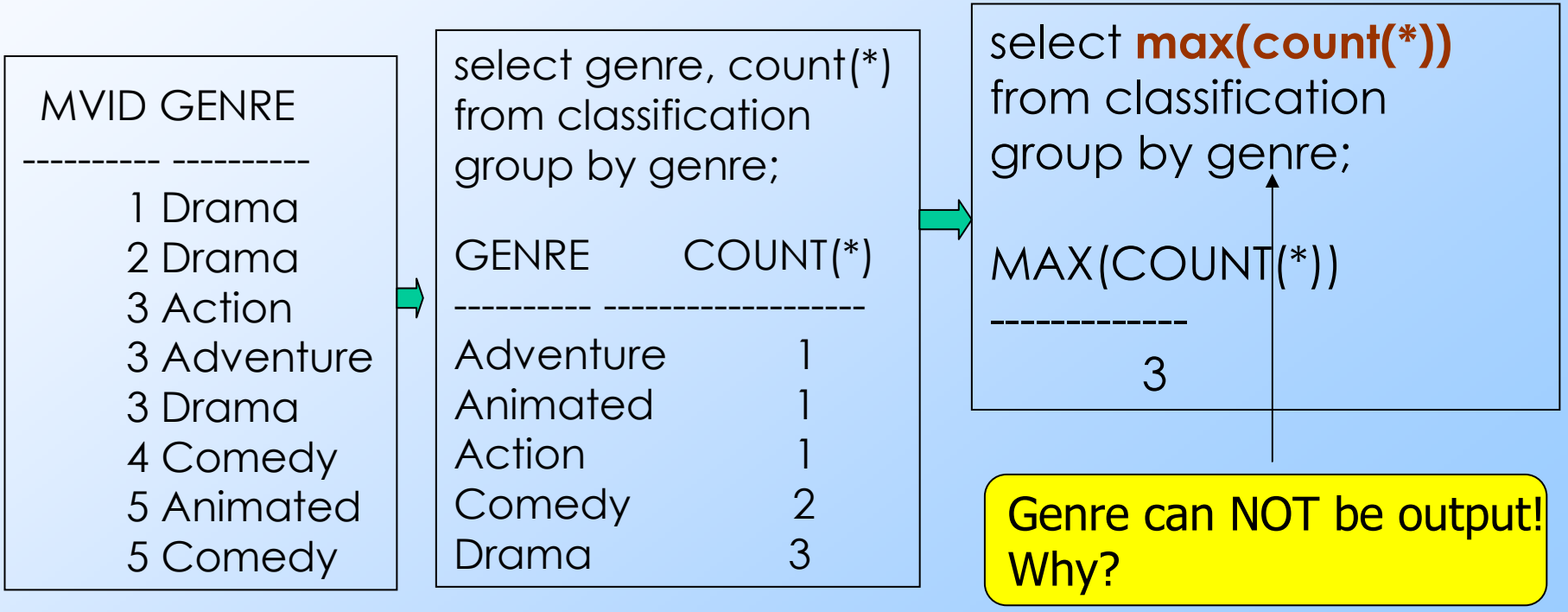
# GROUP BY ...

- With tuples grouped into groups, groups are represented by the group-by attributes, and tuples become unrecognisable.
- Only group-by attributes and aggregates of groups can be output.

```
  1  select genre, mvID
  2  from classification
  3* group by genre
SQL> /
select genre, mvID
              *
ERROR at line 1:
ORA-00979: not a GROUP BY expression
```

# GROUP BY: Nested Aggregation

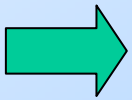
- Aggregates for groups can be further aggregated.



# HAVING: select groups and aggregates

- HAVING specifies conditions for choosing groups and aggregates to output.
- Example: What are the genres that have at least two movies? Output these genres and their total number of movies.

```
 MVID GENRE
--------- ---------
        1 Drama
        2 Drama
        3 Action
        3 Adventure
        3 Drama
        4 Comedy
        5 Animated
        5 Comedy
```



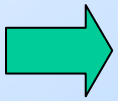
```
select genre, count(*)
from classification
group by genre
having count(*) >=2;

GENRE       COUNT(*)
--------- ------------------
Comedy              2
Drama               3
```

# WHERE vs. HAVING

- A WHERE clause chooses tuples for output or further aggregation. A HAVING clause chooses groups and aggregates for output.
- A WHERE clause must before the GROUP BY and HAVING clauses.
- Example: For genres starting with C or D, output those that have at least 3 movies.

```
 MVID GENRE
--------- ---------
        1 Drama
        2 Drama
        3 Action
        3 Adventure
        3 Drama
        4 Comedy
        5 Animated
        5 Comedy
```



```
select genre, count(*)
from classification
where genre like 'C%' or genre like 'D%'
group by genre
having count(*) >=3

GENRE      COUNT(*)
---------- --------------------
Drama                 3
```

# Oracle SQL/Plus – Pragmatics

- Double vs. single quotes:
  - Double quotes for identifiers – table and column names.
  - Single quotes for strings.
- Example: Which movie has Tom Hanks?

```
SQL> select mvID
  2  from Cast
  3  where actor="Tom Hanks";
where actor="Tom Hanks"
            *
ERROR at line 3:
ORA-00904: "Tom Hanks": invalid identifier
```

# Oracle SQL/Plus – Pragmatics

- Case of letters.
  - Keywords and identifiers are not case sensitive.
    - Many examples have been given throughout the lecture.
  - Strings (in single quotes) are case sensitive.

```
SQL> select *
  2  from Classification
  3  where genre='action';

no rows selected
```

**Classification**

| mvID | Genre |
|---|---|
| 1 | Drama |
| 2 | Drama |
| 3 | Drama |
| 3 | Action |
| 3 | Adventure |
| 4 | Comedy |
| 5 | Comedy |
| 5 | Animated |

# Oracle SQL/Plus – SELECT *Anything*

- Any expression that makes sense can appear as an element of a SELECT clause. Dual is a system dummy table for this purpose.
  - Example: Find the current time and date. Sysdate is a built-in function.

```
Select sysdate
from dual;
```

  - Example: calculate 11*5.

```
Select 11*5
from dual;
```

  - Example: Print the message "Hello World!"

```
Select 'Hello World!'
 from dual;
```

# Oracle SQL/Plus – Data Dictionary

- TAB: list of tables and views for a user.
  select * from tab;
- USER_CONSTRAINTS: constraints on tables.

```
select constraint_name, constraint_type
From user_constraints
Where table_name = 'DIRECT';
```

- USER_CONS_COLUMNS: constraints on columns.

```
Select constraint_name, column_name
From user_cons_columns
Where table_name='DIRECT';
```

More information: The SQL*Plus tutorial by R. Holowczak on Blackboard.

# Summary

- A complete SQL query:

```
SELECT output attributes or aggregates
 FROM  list of tables
WHERE  conditions for selecting tuples
GROUP BY attributes for grouping tuples
HAVING conditions for selecting groups
ORDER BY attributes for ordering output
```

- SQL/Plus Tutorial at Blackboard → Useful URLs.
  - Not all Oracle SQL/Plus programming details can be explained in lecture. Grasp Oracle SQL*Plus programming by studying this tutorial, and other online resources.